Web technologies

PHP Advanced Topics

Lecture slides – Winter term 2010

Alpen-Adria-University Klagenfurt

Christian Koncilia

Building Applications in PHP

Building Applications in PHP

- 10 Years ago, PHP was a simple scripting language
- Used to „hack“ small pieces of logic into HTML
- Nowadays, PHP is a mature object-oriented programming language
- Approaches to build a PHP-App:
  - Use Framework
  - Use PHP Standalone

Patterns

- Important patterns / topics in all programming languages
- MVC
- Layout
- Filters / Validators
- ORM
- Security
- Performance (backend / frontend)
- …

MVC

php_badstyle.php

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="my2style.css" />
  </head>
  <body>
    <?php session_start();
          $pdo = new PDO('mysql:host=localhost;dbname=dbtech', 'root', 'root');
          $stmt = $pdo->prepare("SELECT * FROM products");
          $stmt->execute();
          $dbresult = $stmt->fetchAll(PDO::FETCH_ASSOC);

          if (isset($_SESSION['username'])) {
            echo "Welcome back, user";
          } else {
            echo "You are not loged in";
          }
          foreach ($dbresult as $record) {
            foreach ($record as $k => $v) {
              echo $k . ": " . $v . "<br>";
            }
          }
```

This program sucks! Why?...

MVC

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="my2style.css" />
  </head>
  <body>
    <?php session_start();
          $pdo = new PDO('mysql:host=localhost;dbname=dbtech',
          $stmt = $pdo->prepare("SELECT * FROM products");
          $stmt->execute();
          $dbresult = $stmt->fetchAll(PDO::FETCH_ASSOC);

          if (isset($_SESSION['username'])) {
            echo "Welcome back, user";
          } else {
            echo "You are not loged in";
          }
          foreach ($dbresult as $record) {
            foreach ($record as $k => $v) {
              echo $k . ": " . $v . "<br>";
            }
          }
```

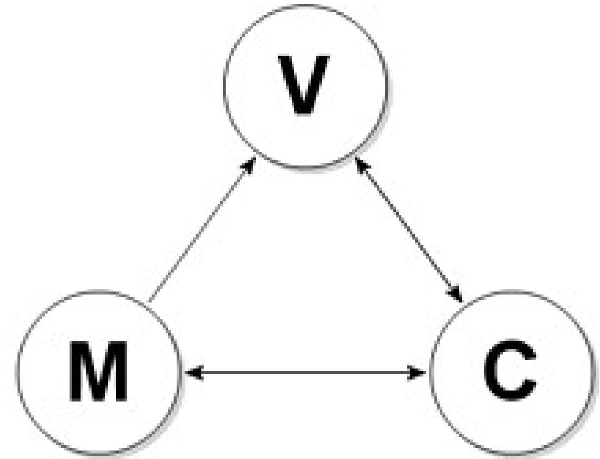
View (Presentation)

Model (Data access)

Controller (Logic)

MVC

- MVC = Model / View / Controller
- MVC means to separate
  - Business Logic & Data access (Model)
  - Presentation (View)
  - Input/Response Handler (Controller)



Model-View-Controller

- Usually, this means to have:
  - One class for the model (reuseable in different controllers)
  - One class for the controller
  - One view

MVC

php_mvc_v.php

- Start with the View:

```
<?php
  include 'php_mvc_c.php'; // Include the Controller
?>

<html>
  <head>
    <link rel="stylesheet" type="text/css" href="my2style.css" />
  </head>
  <body>
    <h1>MVC Example</h1>
    <?php
      echo $welcome_txt;
      foreach ($dbresult as $record) {
         foreach ($record as $k => $v) {
            echo $k . ": " . $v . "<br>";
         }
      }
    ?>
  </body>
</html>
```

MVC

php_mvc_c.php

- Here is the controller:

```
<?php

  include 'php_mvc_m.php'; // Include the Model
  session_start();
  $dbresult = fetchAllProducts();

  if (isset($_SESSION['username'])) {
     $welcome_txt = "Welcome back, user<br><br>";
  } else {
     $welcome_txt = "You are not loged in<br><br>";
  }
```

MVC

php_mvc_m.php

- Here is the model:

```
<?php
function getDBConnection() {
  try {
    $pdo = new PDO('mysql:host=localhost;dbname=dbtech', 'root', 'root');
    return $pdo;
  } catch (Exception $e) {
    throw new Exception ("Sorry, could not connect... " . $e->getMessage());
  }
}

function fetchAllProducts() {
  $pdo = getDBConnection();
  try {
    $stmt = $pdo->prepare("SELECT * FROM products");
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
  } catch (Exception $e) {
    throw new Exception ("Sorry, could not read data... " . $e->getMessage());
  }
}
```

MVC

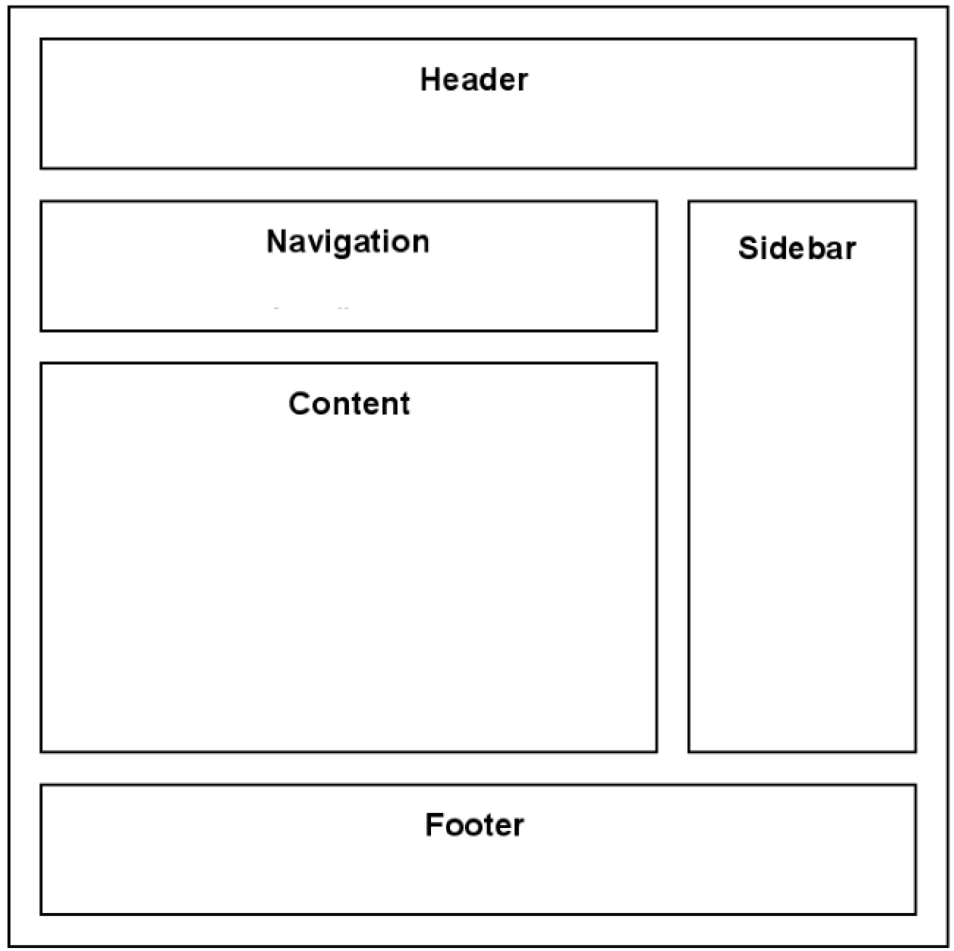
- All the examples presented are non-OO
- It would be even better to use classes

```
Class myDatabase() {
    public function connect() {
        ...
    }
    ...
}

Class ProductsModel extends myDatabase () {
     ...
}
```

Layout

- Usually, all pages of a web app share a basic layout
- For instance:



Layout

- Stupid approach:
  - Implement the header / footer / sidebar / nav-controll / … in each single page

Page1.php

```
<body>
<div>I am the fancy header!</div>
<div>
  <span class="m_enabled">Menu 1</span>
  <span class="m_disabled">Menu 2</span>
</div>
<div>
  Here is the great content of page1
</div>
<div>I am the fancy footer!</div>
</body>
```

Page2.php

```
<body>
<div>I am the fancy header!</div>
<div>
   <span class="m_disabled">Menu 1</span>
   <span class="m_enabled">Menu 2</span>
</div>
<div>
  Here is the great content of page 2
</div>
<div>I am the fancy footer!</div>
</body>
```

Layout

- Better approach:
  - Implement the header / footer / sidebar / nav-controll / … in single php scripts
- [All examples in \layout folder]

Layout - Header

header.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
       <head>
              <title>Hello World Demonstration Document</title>
              <link rel="stylesheet" type="text/css" href="../my2style.css" />
              <link rel="stylesheet" type="text/css" href="layoutexample.css" />
       </head>
       <body>

       <div id="header">I am the fancy header...</div>
```

Layout – Navigation Area

navbar.php

```
<div id="navbar">
<?php
$currentfile = basename($_SERVER['PHP_SELF']);
switch ($currentfile) {
  case "page2.php":
       echo "<a href='page1.php'><span class='m_enabled'>Page1</span></a>";
       echo "<span class='m_disabled'>Page2</span>";
       break;
  case "page1.php":
       echo "<span class='m_disabled'>Page1</span>";
       echo "<a href='page2.php'><span class='m_enabled'>Page2</span></a>";
       break;
  default: throw new Exception("Oops... an error occured");
}
?>
</div>
```

Layout – Page1 / Page2

page1.php / page2.php

```
<?php
  include 'header.php'; // Include the Header
  include 'navbar.php'; // Include the Navigation Area
?>

<div id="content">
This is the content on page1... This is the content on page1... This is the content
on page1... This is the content on page1... This is the content on page1...
This is the content on page1... This is the content on page1... This is the content
on page1... This is the content on page1... This is the content on page1...
</div>

<?php
  include 'sidebar.php'; // Include the Sidebar
  include 'footer.php';  // Include the Footer
?>
```

Layout – Sidebar

sidebar.php

```
<div id="sidebar">
	This is your fancy sidebar... This is your fancy sidebar... This is your fancy sidebar... This is your fancy sidebar... This is your fancy sidebar...
</div>
```

Layout – Footer

footer.php

```
<div id="footer">I am the fancy footer...</div>
</body>
</html>
```

Filters / Validators

- Never ever trust the user (see also security)
- Two basic rules
  - Validate every single input
  - Filter every single input
- Validators: return true/false
  - is_int("23") -> true
  - is_int("abc") -> false
- Filters: return a (filtered) part of the input
  - intval("23") -> 23
  - intval("abc") -> 0

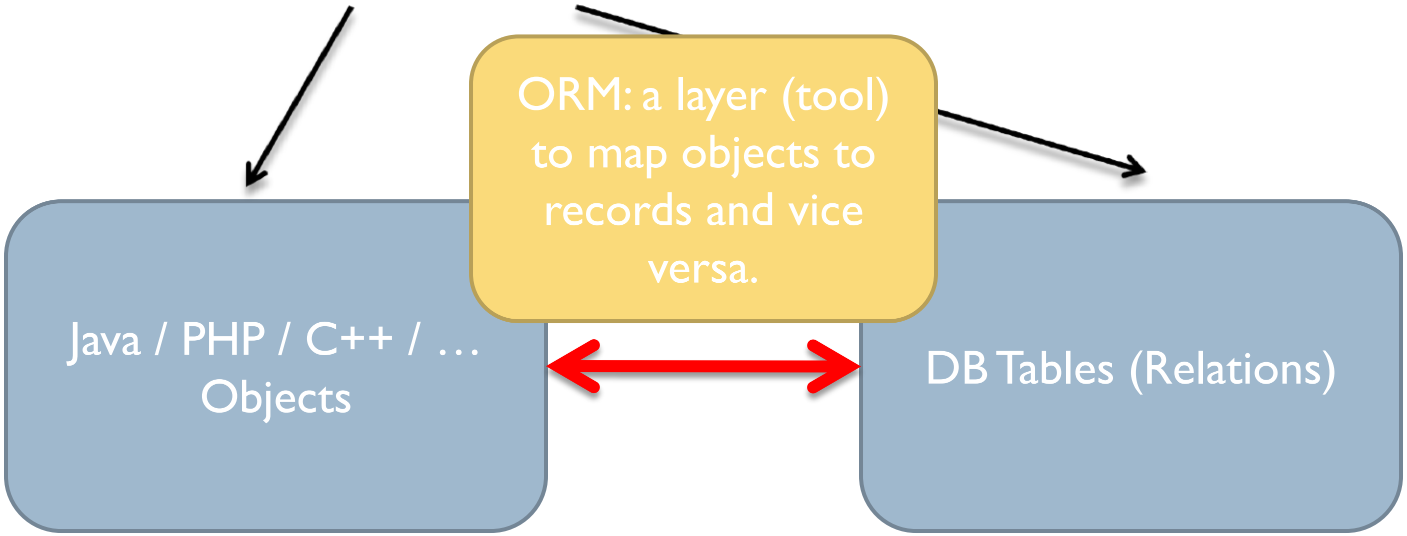
Filters / Validators

php_preg.php

- Often filters & validators are more complex
- E.g.: check if email address is valid
- Use regular expressions
  - preg_filter
  - preg_match
  - preg_replace
  - …
- Implement your own filter & validator classes

ORM

- ORM = Object Relational Mapping



- Inheritance
- Aggregation
- Interfaces
- Overrides
- ...

- Tables
- Records
- Keys
- Index
- Trigger
- ...

„ORM“ or not ORM

- ORM sucks
  - Need to learn something new
  - Overhead in my applications
  - Slower
  - Produces inefficient SQL
- ORM rulz
  - ORM saves time
  - time = money
  - Makes maintenance easy (both, at the DB or OOPL level)
  - Creates a „virtual object oriented database“ or „virtual database of objects“

ORM Tools

- PHP
  - Doctrine
  - Propel
  - Xyster_ORM (part of Xyster)
  - Zend_DB (part of Zend Framework)
  - CakePHP
  - PdoMap
  - Rocks
  - …

ORM: YAML Example

```yaml
---
detect_relations: true
Order:
   connection: 0
   tableName: IORDER
   columns:
     id: integer(4)
     client_name: string(255)
     client_address: string(255)
Item:
   connection: 0
   tableName: ITEM
   columns:
     id:
       type: integer(4)
       fixed: false
       unsigned: false
       primary: true
       autoincrement: true
     title: string(255)
     cost: float
     order_id: integer
```

- YAML: Yet Another Markup Language
- Plattform independent schema definitions
- Doctrine will generate
  - All DB Tables
  - All PHP classes
- Execute this file

```
php doctrine build-all-reload
```

ORM: YAML Example: Tables generated

- Doctrine will automatically generate a DB with the following tables

```
CREATE TABLE  `doctrinetest`.`iorder` (
`id` bigint(20) NOT NULL AUTO_INCREMENT,
`client_name` varchar(255) DEFAULT NULL,
`client_address` varchar(255) DEFAULT
   NULL,
PRIMARY KEY (`id`))
ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE  `doctrinetest`.`item` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`title` varchar(255) DEFAULT NULL,
`cost` float(18,2) DEFAULT NULL,
`order_id` bigint(20) DEFAULT NULL,
   PRIMARY KEY (`id`),  KEY
`order_id_idx` (`order_id`),
CONSTRAINT `ITEM_order_id_IORDER_id`
   FOREIGN KEY (`order_id`) REFERENCES
   `iorder` (`id`))
ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

ORM:YAML Example: Classes generated

- Doctrine will generate all tabels necessary

- Order-Classes:
  - BaseOrder.php: Used by doctrine to define contents of tables („interface“ to the DB)
  - OrderTable.php: Represents the DB table
  - Order.php: The class you will use

```
abstract class BaseOrder extends Doctrine_Record
{
    public function setTableDefinition()
    {
        $this->setTableName('IORDER');
        $this->hasColumn('client_name', 'string',
                          255,
             array(
            'type' => 'string',
            'length' => '255',
            ));
        $this->hasColumn('client_address',
                          'string', 255,
            array(
            'type' => 'string',
            'length' => '255',
            ));
    }

    public function setUp()
    {
        parent::setUp();
        $this->hasMany('Item', array(
            'local' => 'id',
            'foreign' => 'order_id'));
    }
}
```

YAML Example: Classes generated

- Order.php

```
class Order extends BaseOrder
{
    // put your methods here…
}
```

CRUD Example

- Straight forward to implement logic without caring about how to implement CRUD
- Example: Create

```
// Create an order and set its vars
$order = new Order();
$order->client_name = "C. Koncilia";
$order->client_address = "9020";

// Add a new item
$order[0]->title = "Vodka";
$order[0]->cost = 15;

// Add another item
$order[1]->title = "Good pickles";
$order[1]->cost = 2;

// Save the order (and all items)
// to the database
$order->save();
```

Btw, CRUD = Create, Read, Update, Delete

CRUD Example

- Example: Read

```
// Create an order object
$o = new Order();

// get data for order #1782
$o = Doctrine::getTable("order")
   ->find(1782);

print_r $o;
```

CRUD Example

- Example: Update

```
// Create an order object
$o = new Order();

// get data for order #1782
$o = Doctrine::getTable("order")
   ->find(1782);

// set new values
$o->client_name = "C. Koncilia";

// save new values
$o->save();
```

CRUD Example

- Example: Delete

```
// Create an order object
$o = new Order();

// get data for order #1782
$o = Doctrine::getTable("order")
   ->find(1782);

// delete record
$o->delete();
```

Security

- Rules discussed:
  - Use session cookies (don‘t use URLs)
  - Use filters & validators everywhere!
- Security issues:
  - XSS (Cross Side Scripting): Attack a client
  - SQLi (SQL Injection): Attack a server
  - CSRF (Cross Side Request Forgery): Highjack requests
  - …

Security (SQLi)

php_sqli.php

- How SQL injection works… (see sample code)

Security

- Enter "`1 or true`" and click submit
- Enter "`1; drop table products`" and click submit
- Now try to enter "`1`" again

Security

php_sqli_safe.php

- Avoid SQLi: Filter / Validate data!

```
// a simple filter
$id = intval($vars['prodid']);

// a simple validation
if ($vars['prodid'] != strval($id)) {
   echo '<strong style="color: #ff0000;">Not a valid ID</strong>';
}
else {
$pdo = new PDO(
  'mysql:host=localhost;dbname=dbtech', 'root', 'root,
);
$stmt = $pdo->prepare("SELECT * FROM products WHERE id = " . $id);
…
```

Security (XSS)

php_htmli.php

- Cross Side Scripting = HTML injection
- Basic idea: if server takes user input and returns the input to the user (unfiltered)…
- E.g., a search form: after clicking on submit, we return the result + inform the user about what he/she was looking for

```
$vars = $_POST;
if (array_key_exists('search', $vars)) {
   echo "You are searching fore: " . $vars['search'] . "<br><br>";

   // do the search here...
}
```

Security (XSS)

- Enter a search string and click submit
- Enter "`<b>string</b>`" and click submit
- Enter "`<script type="text/javascript">alert("XSS")</script>`" and click submit

Security (XSS)

php_htmli_safe.php

- Avoid CSS: never ever return user input un-escaped

```
$vars = $_POST;
if (array_key_exists('search', $vars)) {
echo "You are searching fore: " . htmlentities($vars['search']) . "<br><br>";

// do the search here...

}
```

Performance

- Server side:
  - Write better code (e.g., use Zend Profiler)
  - Create better databases (analyse queries – see DB-Tech LV)
  - Use better hardware (cheapest „solution“)
- Client side:
  - Install „YSlow“ plugin for FireFox
  - Analyse your website using YSlow

Appendix

Production vs. Development

- In Production mode
  - Turn off all PHP errors
  - Turn off all DB errors
  - Use try-catch to catch errors and
    - show something like „Sorry, we ran into a technical problem“
    - Log error in details / Send email to admin
- In Development mode
  - Turn on all PHP errors and warnings
  - Turn on all DB errors and warnings

Production vs. Development

- Use something like this in a code that gets called at each request (e.g. header.php):

```
if (in_array($_SERVER['HTTP_HOST'],
    array('localhost','dev.webtech2010.com')))
{
   define('APPLICATION_ENVIRONMENT',  'development');
   $pdo->setAttribute( PDO::ATTR_ERRMODE,
                       PDO::ERRMODE_EXCEPTION );
   // Report all errors except E_NOTICE
   error_reporting(E_ALL ^ E_NOTICE);
} else {
   define('APPLICATION_ENVIRONMENT', 'production');
   error_reporting(0);
}
```

Production vs. Development

- Later on use something like:

```
if ('production' == APPLICATION_ENVIRONMENT)
{
   ...
}
```